2
Drilling Deep into Process Management, Job Control, and Automation

In the last chapter, we introduced ourselves to the Bash shell environment in Linux. You learned basic commands and wrote your first shell script as well.

You also learned about process management and job control. This information will be very useful for system administrators in automation and in terms of solving many problems.

In this chapter, we will cover the following topics:

- Monitoring processes with `ps`
- Job management–working with `fg`, `bg`, `jobs`, and `kill`
- Exploring `at` and `crontab`

Introducing process basics

A running instance of a program is called a process. A program stored in the hard disk or pen drive is not a process. When that stored program starts executing, then we say that process has been created and is running.

Let's very briefly understand the Linux operating system boot-up sequence:

1. In PCs, initially, the BIOS chip initializes system hardware, such as PCI bus, and display device drivers.
2. Then the BIOS executes the boot loader program.

3. The boot loader program then copies the kernel in the memory and, after basic checks, it calls a kernel function `start_kernel()`.
4. The kernel then initializes the OS and creates the first process called `init`.
5. You can check the presence of this process with the following command:

```
$ ps -ef
```

6. Every process in the OS has one numerical identification associated with it. It is called a `process ID`. The process ID of the `init` process is `1`. This process is the parent process of all user space processes.
7. In the Linux OS, every new process is created by a system call called `fork()`.
8. Therefore, every process has a process ID, as well as the parent process ID.
9. We can see the complete process tree using the following command:

```
$ pstree
```

You can see the very first process as `init`, as well as all other processes with a complete parent and child relation between them. If we use the `$ps -ef` command, then we can see that the `init` process is owned by the root and its parent process ID is `0`. This means that there is no parent for `init`:

```
[student@localhost ~]$ pstree
systemd─┬─ModemManager───2*[{ModemManager}]
        ├─NetworkManager───2*[{NetworkManager}]
        ├─VGAuthService
        ├─abrt-dbus───2*[{abrt-dbus}]
        ├─2*[abrt-watch-log]
        ├─abrtd
        ├─accounts-daemon───2*[{accounts-daemon}]
        ├─alsactl
        ├─at-spi-bus-laun─┬─dbus-daemon───{dbus-daemon}
        │                 └─3*[{at-spi-bus-laun}]
        ├─at-spi2-registr───2*[{at-spi2-registr}]
        ├─atd
        ├─auditd─┬─audispd─┬─sedispatch
        │        │         └─{audispd}
        │        └─{auditd}
        ├─avahi-daemon───avahi-daemon
        ├─chronyd
        ├─colord───2*[{colord}]
        ├─crond
        ├─cupsd
        ├─2*[dbus-daemon───{dbus-daemon}]
        ├─dbus-launch
        ├─dconf-service───2*[{dconf-service}]
```

Therefore, with the exception of the `init` process, all other processes are created by some other process. The `init` process is created by the kernel itself.

The following are the different types of processes:

- **Orphan process**: If, by some chance, the parent process is terminated, then the child process becomes an orphan process. The process that created the parent process, such as the grandparent process, becomes the parent of the orphan child process. As a last resort, the `init` process becomes the parent of the orphan process.
- **Zombie process**: Every process has one data structure called the process control table. This is maintained in the operating system. This table contains information about all the child processes created by the parent process. If, by chance, the parent process is sleeping or is suspended due to some reason or other and the child process is terminated, then the parent process cannot receive the information about the child process termination. In such cases, the child process that has been terminated is called the zombie process. When the parent process awakes, it will receive a signal regarding the child process termination and the process control block data structure will be updated. The child process termination is then completed.
- **Daemon process**: Until now, we have started every new process in a Bash Terminal. Therefore, if we print any text with the `$ echo` command, it will be printed in the Terminal itself. There are certain processes that are not associated with any Terminal. Such a process is called a daemon process. These processes are running in the background. An advantage of the daemon process is that it is immune to the changes happening to the Bash shell that has created it. When we want to run certain background processes, such as a DHCP server, then the daemon process is very useful.

Monitoring processes using ps

We have used the `ps` command in the introduction. Let's learn more about it:

- To list the processes associated with our current Bash shell Terminal, enter the following command:

```
$ ps
```

```
[student@localhost ~]$
[student@localhost ~]$ ps
   PID TTY          TIME CMD
  7853 pts/0    00:00:00 bash
  8546 pts/0    00:00:00 ps
[student@localhost ~]$
[student@localhost ~]$
```

- To list processes, along with the parent process ID associated with the current Terminal, enter the following command:

```
$ ps -f
```

```
[student@localhost ~]$
[student@localhost ~]$
[student@localhost ~]$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
student     7853    7846  0 13:19 pts/0    00:00:00 bash
student     8673    7853  0 14:04 pts/0    00:00:00 ps -f
[student@localhost ~]$
[student@localhost ~]$
```

- We can see the process ID in the `PID` column and the parent process ID, in the `PPID` column in the preceding output.

- To list processes with the parent process ID along with the process state, enter the following command:

```
$ ps -lf
```

```
[student@localhost ~]$
[student@localhost ~]$
[student@localhost ~]$ ps -lf
F S UID          PID    PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S student     7853    7846  0  80   0 - 29174 do_wai 13:19 pts/0    00:00:00 bas
0 R student     8715    7853  0  80   0 - 37766 -      14:06 pts/0    00:00:00 ps
[student@localhost ~]$
[student@localhost ~]$
[student@localhost ~]$
```

- In the preceding output, the column with `S` (state) shows the current state of a process, such as `R` for running and `S` for suspended state.
- To list all the processes running in the operating system, including the system processes, enter the following command:

```
$ ps -ef
```

```
[student@localhost ~]$
[student@localhost ~]$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 10:16 ?        00:00:04 /usr/lib/systemd/systemd --switched-root
root           2       0  0 10:16 ?        00:00:00 [kthreadd]
root           3       2  0 10:16 ?        00:00:00 [ksoftirqd/0]
root           5       2  0 10:16 ?        00:00:00 [kworker/0:0H]
root           7       2  0 10:16 ?        00:00:00 [migration/0]
root           8       2  0 10:16 ?        00:00:00 [rcu_bh]
root           9       2  0 10:16 ?        00:00:01 [rcu_sched]
root          10       2  0 10:16 ?        00:00:00 [watchdog/0]
root          12       2  0 10:16 ?        00:00:00 [kdevtmpfs]
root          13       2  0 10:16 ?        00:00:00 [netns]
root          14       2  0 10:16 ?        00:00:00 [khungtaskd]
root          15       2  0 10:16 ?        00:00:00 [writeback]
root          16       2  0 10:16 ?        00:00:00 [kintegrityd]
root          17       2  0 10:16 ?        00:00:00 [bioset]
root          18       2  0 10:16 ?        00:00:00 [kblockd]
root          19       2  0 10:16 ?        00:00:00 [md]
root          25       2  0 10:16 ?        00:00:00 [kswapd0]
root          26       2  0 10:16 ?        00:00:00 [ksmd]
root          27       2  0 10:16 ?        00:00:00 [khugepaged]
root          28       2  0 10:16 ?        00:00:00 [crypto]
root          36       2  0 10:16 ?        00:00:00 [kthrotld]
root          38       2  0 10:16 ?        00:00:00 [kmpath_rdacd]
root          39       2  0 10:16 ?        00:00:00 [kpsmoused]
```

- The process names in `[]` are kernel threads. If you are interested in more options for the `ps` command, you can use the following command:

```
$ man ps
```

- To find a particular process, you can use the following command:

```
$ ps -ef | grep "process_name"
```

- The command with `grep` will display the process with `process_name`.

- If we want to terminate the running process, enter the following command:

```
$ kill  pid_of_process_to_be_killed
```

```
[student@localhost ~]$
[student@localhost ~]$ ps
   PID TTY          TIME CMD
  9508 pts/0    00:00:00 bash
  9555 pts/0    00:00:00 sleep
  9575 pts/0    00:00:00 ps
[student@localhost ~]$
[student@localhost ~]$ kill 9555
[1]+  Terminated              sleep 10000
[student@localhost ~]$
[student@localhost ~]$ ps
   PID TTY          TIME CMD
  9508 pts/0    00:00:00 bash
  9609 pts/0    00:00:00 ps
[student@localhost ~]$
```

- Many a time, if the process is not killed by the `$ kill` command, you may need to pass additional options to ensure that the required process is killed, which is shown as follows:

```
$ kill -9 pid_of_process_to_be_killed
```

- We can terminate the process with the name of a process, instead of using the process ID, as follows:

```
$ pkill command_name
$ pkill sleep
```

- Or:

```
$ pkill  -9  command_name
```

```
[student@localhost ~]$
[student@localhost ~]$ ps
   PID TTY          TIME CMD
  3089 pts/0    00:00:00 bash
  3305 pts/0    00:00:00 sleep
  3318 pts/0    00:00:00 ps
[student@localhost ~]$
[student@localhost ~]$ pkill sleep
pkill: killing pid 3298 failed: Operation not permitted
[1]+  Terminated              sleep 10000
[student@localhost ~]$
[student@localhost ~]$ ps
   PID TTY          TIME CMD
  3089 pts/0    00:00:00 bash
  3344 pts/0    00:00:00 ps
[student@localhost ~]$
[student@localhost ~]$
```

- To know more about various flags of `kill`, enter the following command:

```
$ kill -l
```

- This displays all the signals or software interrupts used by the operating system. When we enter the `$ kill` command, the operating system sends the `SIGTERM` signal to the process.
- If the process is not killed by this command, then we enter the following command:

```
$ kill -9 process_name
```

- This sends `SIGKILL` to the process to be killed.

Process management

Since we have understood the command to check processes, we will learn more about managing different processes.

- In a Bash shell, when we enter any command or start any program, it starts running in the foreground. In such a situation, we cannot run more than one command in the foreground. We need to create many Terminal windows for starting many processes. If we need to start many processes or programs from the same Terminal, then we will need to start them as background processes.
- If we want to start a process in the background, then we need to append the command in the Bash shell by `&`.
- If I want to start my `Hello` program as the background process, then the command would be as follows:

```
$ Hello &
```

- If we terminate any command by `&`, then it starts running as the background process.

For example, we will issue a simple `sleep` command, which creates a new process. This process sleeps for the duration, which is mentioned in the integer value next to the `sleep` command:

1. The following command will make the process sleep for 10,000 seconds. This means we will not be able to run any other command from the same Terminal:

```
$ sleep 10000
```

2. Now, you can press the *Ctrl* + *C* key combination to terminate the process created by the `sleep` command.

```
[student@localhost ~]$
[student@localhost ~]$ ps
   PID TTY          TIME CMD
  3089 pts/0    00:00:00 bash
  3729 pts/0    00:00:00 ps
[student@localhost ~]$
[student@localhost ~]$
[student@localhost ~]$ sleep 10000

^C
[student@localhost ~]$
[student@localhost ~]$
```

3. Now, use the following command:

```
$ sleep 10000 &
```

   The preceding command will create a new process, which will be put to sleep for `10000` seconds; but this time, it will start running in the background. Therefore, we will be able to enter the next command in the Bash Terminal.

4. Since the newly created process is running in the background, we can enter new commands very easily in the same Terminal window:

```
$ sleep 20000 &
$ sleep 30000 &
$ sleep 40000 &
```

5. To check the presence of all the processes, enter the following command:

```
$ jobs
```

```
[student@localhost ~]$
[student@localhost ~]$ sleep 10000 &
[1] 4419
[student@localhost ~]$ sleep 20000 &
[2] 4426
[student@localhost ~]$ sleep 30000 &
[3] 4433
[student@localhost ~]$ sleep 40000 &
[4] 4440
[student@localhost ~]$ jobs
[1]   Running                 sleep 10000 &
[2]   Running                 sleep 20000 &
[3]-  Running                 sleep 30000 &
[4]+  Running                 sleep 40000 &
[student@localhost ~]$
[student@localhost ~]$
```

The `jobs` command lists all the processes running in the Terminal, including foreground and background processes. You can clearly see their status as running, suspended, or stopped. The numbers in `[]` show the job ID. The `+` sign indicates which command will receive `fg` and `bg` commands by default. We will study them in the following topics.

6. If you want to make any existing background process run in the foreground, then use the following command:

```
$ fg 3
```

The preceding command will make the job number `3` run in the foreground instead of the background.

If we want to make the process stop executing and get it suspended, then press *Ctrl* + Z. This key combination makes the foreground process stop executing. Please note that the process has stopped, but is not terminated.

```
[student@localhost ~]$ fg 3
sleep 30000

^Z
[3]+  Stopped                 sleep 30000
[student@localhost ~]$
[student@localhost ~]$ jobs
[1]   Running                 sleep 10000 &
[2]   Running                 sleep 20000 &
[3]+  Stopped                 sleep 30000
[4]-  Running                 sleep 40000 &
[student@localhost ~]$
```

7. To make the stopped process continue running in the background, use the following command:

```
$ bg job_number
$ bg 3
```

   The preceding command will make suspended job process number `3` run in the background.

8. If you wish to terminate the process, you can use the job ID or process ID as follows:

```
$ jobs -l          //  This will list jobs with pid
$ kill pid         // or
$ kill %job_id     // This will kill job
$ kill %3
```

Process monitoring tools – top, iostat, and vmstat

We can view the native performance of various processes in an OS by using the following tools:

- To view a dynamic real-time view of the running processes in an OS, use the following command:

```
$ top
```

```
top - 16:40:34 up  1:07,  2 users,  load average: 0.25, 0.12, 0.08
Tasks: 180 total,   1 running, 178 sleeping,   1 stopped,   0 zombie
%Cpu(s):  1.4 us,  0.3 sy,  0.0 ni, 98.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  1867024 total,   615620 free,   726984 used,   524420 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.   925896 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 2426 student   20   0 1890548 191072  48088 S  2.0 10.2   0:18.09 gnome-shell
 1764 root      20   0  289548  32460  10200 S  1.0  1.7   0:04.18 X
 4345 student   20   0  708344  22420  14360 S  0.7  1.2   0:00.56 gnome-terminal-
  401 root      20   0       0      0      0 S  0.3  0.0   0:00.72 xfsaild/dm-0
 1099 root      20   0  231348   6104   4752 S  0.3  0.3   0:04.14 vmtoolsd
 1489 root      20   0  562344  16588   5880 S  0.3  0.9   0:00.65 tuned
 4587 student   20   0  157716   2236   1532 R  0.3  0.1   0:00.07 top
    1 root      20   0  128436   7232   4064 S  0.0  0.4   0:01.90 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S  0.0  0.0   0:00.11 ksoftirqd/0
    5 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 kworker/0:0H
    7 root      rt   0       0      0      0 S  0.0  0.0   0:00.00 migration/0
    8 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcu_bh
    9 root      20   0       0      0      0 S  0.0  0.0   0:00.41 rcu_sched
   10 root      rt   0       0      0      0 S  0.0  0.0   0:00.02 watchdog/0
   12 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kdevtmpfs
   13 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 netns
   14 root      20   0       0      0      0 S  0.0  0.0   0:00.00 khungtaskd
   15 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 writeback
   16 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 kintegrityd
   17 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 bioset
   18 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 kblockd
```

An explanation of the `top` command generated output is as follows:

The `$top` command displays a lot of information about the running system.

The first line of the display is shown as follows:

```
top - 16:53:10 up  1:19,  2 users,  load average: 0.17, 0.13, 0.09
```

The description of fields in the first line is as follows:

- Current time
- System uptime
- Number of users logged in
- Load average of `5`, `10`, and `15` minutes, respectively

The second line is shown as follows:

```
Tasks: 181 total,   1 running, 179 sleeping,   1 stopped,   0 zombie
```

This line shows the summary of tasks or processes. It shows the total number of all the processes, which includes the total number of running, sleeping, stopped, and zombie processes. The third line is shown as follows:

```
%Cpu(s):  0.3 us,  0.0 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
```

This line shows information about CPU usage as a `%` in different modes as follows:

- `* us (user)`: CPU usage in `%` for running (un-niced) user processes
- `* sy (system)`: CPU usage in `%` for running kernel processes
- `* ni (niced)`: CPU usage in `%` for running niced user processes
- `* wa (IO wait)`: CPU usage in `%` waiting for IO completion
- `* hi (hardware interrupts)`: CPU usage in `%` for serving hardware interrupts
- `* si (software interrupts)`: CPU usage in `%` for serving software interrupts
- `* st (time stolen)`: CPU usage in `%` for time stolen for this VM by the hypervisor

The fourth line is shown as follows:

```
KiB Mem :  1867024 total,   612492 free,   729876 used,   524656 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.   922884 avail Mem
```

This line provides information about memory usage. It shows the physical memory that is used, free, available, and used for buffers. The next line shows the swap memory that is available, used, free, and cached.

After this line, we see the table of values with the following columns:

- `PID`: This is the ID of the process
- `USER`: This is the user that is the owner of the process
- `PR`: This is the priority of the process
- `NI`: This is the `NICE` value of the process
- `VIRT`: This is the virtual memory used by the process
- `RES`: This is the physical memory used for the process

- `SHR`: This is the shared memory of the process
- `S`: This indicates the status of the process: `S` = sleep, `R` = running, and `Z` = zombie (S)
- `%CPU`: This is the `%` of CPU used by this process
- `%MEM`: This is the `%` of RAM used by the process
- `TIME+`: This is the total time of activity of this process
- `COMMAND`: This is the name of the process

Let's take a look at the performance monitoring tools `iostat`, `vmstat`, and `sar`:

- To view the statistics of the CPU and the input/output device's utilization, use the following command:

```
$ iostat
```

```
[student@localhost ~]$ iostat
Linux 3.10.0-693.el7.x86_64 (localhost.localdomain)     Thursday 28 December 2017       _x86_64_        (1 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.74    0.48    1.27    1.34    0.00   96.16

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               9.55       554.05         9.86    1333552      23732
scd0              0.01         0.44         0.00       1050          0
dm-0              9.11       539.71         9.00    1299043      21664
dm-1              0.04         0.93         0.00       2228          0

[student@localhost ~]$
```

```
$ iostat -c
```

- Shows only CPU statistics

```
$ iostat -d
```

- Shows only disk statistics
- To view the virtual memory statistics, use the following command:

```
$vmstat
```

```
[student@localhost ~]$ vmstat
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 3  0     12  82260    208 955868    0    0   440    13  122  203  1  1 97  1  0
[student@localhost ~]$
```

```
$ vmstat -s
```

- This shows various event counters and memory statistics

```
$ vmstat -t 1 5
```

- Runs for every one second stop after executing for five intervals

```
$ sar -u 2 3
```

- This will show the CPU activity report three times every 2 seconds:

```
[student@localhost ~]$ sar -u 2 3
Linux 3.10.0-693.el7.x86_64 (localhost.localdomain)     Thursday 28 December 2017  _x86_64_ (1 CPU)

12:16:07  IST     CPU     %user     %nice   %syslem   %iowail    %sleal     %idle
12:16:09  IST     all      1.04      0.00      1.55      0.00      0.00     97.41
12:16:11  IST     all      1.03      0.00      0.00      0.00      0.00     98.97
12:16:13  IST     all      1.03      0.00      0.52      0.00      0.00     98.45
Average:        all      1.03      0.00      0.69      0.00      0.00     98.28
[student@localhost ~]$
```

Understanding "at"

Many a time, we need to schedule a task for a future time, say in the evening at 8 p.m. on a specific day. We can use the `at` command in such a situation.

Sometimes, we need to repeat the same task at a specific time, periodically, every day, or every month. In such situations, we can use the `crontab` command.

Let's learn more about the use of the `at` command. To use the `at` command, the syntax is as follows:

```
$ at time date
```

The following are examples of the `at` command:

- The *Ctrl + D* command will save the `at` job. The task will be executed at 11.15 A.M. This command will log messages to the `log.txt` file at 11.15 a.m.:

```
$ at 11.15 AM
at >  echo "Hello World" > $HOME/log.txt
at >  Control + D
```

- The following command will send an email on March 31, 2015, at 10 A.M.:

```
$ at 10am mar 31 2015
at> echo "taxes due" | mail jon
at> ^D
```

- The following command will make the task run on May 20 at 11 A.M.:

```
$ at 11 am may 20
```

- All the jobs that are scheduled by the `at` command can be listed using the following command:

```
$ atq
```

- To remove a specific job listed by the `atq` command, we can use the following command:

```
$ atrm  job-id
```

Understanding crontab

If we need to run a specific task repetitively, then the solution is to use `crontab`. The syntax of the command is as follows:

```
$ crontab -e
```

This will open a new editor. The following diagram is the syntax to add tasks. The fields to use for repeating tasks at a particular time are explained here:

Finally, to save the jobs, use the following:

```
Press Esc then type :wq
```

The preceding operations will save the job and quit crontab.

The following are a few examples of the `crontab` command:

- Use the following command to run a script every hour at the fifth minute, every day:

```
5 * * * *       $HOME/bin/daily.job >> $HOME/tmp/out  2>&1
```

- Use the following command to run 5 minutes after midnight every day:

```
5 0 * * *       $HOME/bin/daily.job >> $HOME/tmp/out  2>&1
```

- Use the following command to run at 2.15 p.m. on the first of every month–the output is mailed to Paul:

```
15 14 1 * * *      $HOME/bin/monthly
```

- Use the following command to run at 10 P.M. on weekdays, and send the email to `ganesh@abc.com`:

```
0 22 * *  1-5   sendmail ganesh@abc.com  < ~/work/email.txt
```

- The `sendmail` utility is used for sending emails. We can also use the mail utility as follows:

```
sendmail user@example.com  < /tmp/email.txt
```

- The following commands are self-explanatory from the text of the `echo` command:

```
    23 0-23/2  *  *  *  echo "run 23 minutes after midn, 2 am, 4 am,
everyday"
    5  4  *  *  sun    echo "run at 5 minutes after 4 am every Sunday"
```

The following are a few more `crontab` command examples:

Min	Hour	Day / month	Month	Day / week	Execution time
45	0	5	1,6,12	*	00:45 hrs on the fifth day of January, June, and December.
0	18	*	10	1-5	6.00 P.M. every weekday (Monday-Friday), only in October.
0	0	1,10,15	*	*	Midnight on the first, tenth, and fifteenth days of the month.
5,10	0	10	*	1	At 12.05 and 12.10 every Monday, and on the tenth day of every month.

We can add `macros` in the `crontab` file. Use the following to restart `my_program` after each reboot:

```
    @reboot  /bin/my_program
    @reboot echo `hostname` was rebooted at `date` | mail -s "Reboot
notification" ganesh.admin@some-corp.com
```

The following is a summary of a few more macros:

Entry	Description	Equivalent To
`@reboot`	Run once at start-up	None
`@weekly`	Run once a week	`0 0 * * 0`
`@daily`	Run once a day	`0 0 * * *`
`@midnight`	(same as @daily)	`0 0 * * *`
`@hourly`	Run once an hour	`0 * * * *`

Summary

In this chapter, we studied basic process management. You learned about the `ps` command. Using commands such as `jobs`, `fg`, `bg`, `kill`, and `pkill`, we studied job management. Later on, you learned about the `top`, `iostat`, and `vmstat` process monitoring tools.

In the next chapter, you will learn about standard input/output, various meta–characters, and text filters used in shell scripting.